UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aplicación de realidad aumentada multi-plano mediante correspondencias entre puntos de interés.

Alejandro Núñez Valle.
Tutor: Marcos Escudero Viñolo.
Ponente: Jesús Bescós Cano.

Junio 2017

Aplicación de realidad aumentada multi-plano mediante correspondencias entre puntos de interés.

Alejandro Núñez Valle
Tutor: Marcos Escudero Viñolo
Ponente: Jesús Bescós Cano

Video Processing and Understanding Lab
Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2017

Trabajo parcialmente financiado por el Ministerio de Economía y Competitividad del Gobierno de España bajo el proyecto TEC2014-53176-R (HAVideo) (2015-2017)

Resumen

El objetivo de este trabajo fin de grado es la creación de un sistema de realidad aumentada. Mediante el uso de herramientas y conceptos de visión por ordenador, aprendizaje automático y el uso de gráficos, se creará una aplicación que será capaz de detectar un objeto y a partir de este generar realidad aumentada.

Se ha investigado sobre las diferentes posibilidades de creación de aplicaciones de realidad aumentada y se ha optado por utilizar una técnica sin marcadores (*Markerless*). Para realizar nuestro objetivo, el diseño de la aplicación ha tenido diferentes fases: la creación de puntos de interés para la detección y descripción de objetos, el entrenamiento de una máquina de vector soporte, el empleo de técnicas de visión por ordenador y el uso de un motor gráfico. Se ha evaluado el modelo y los resultados son, en general, favorables aunque muestran limitaciones a la hora de detectar el objeto. Como conclusión podemos afirmar que, a pesar de las limitaciones, los resultados de la aplicación desarrollada son positivos, por lo que con mayor trabajo sobre esta versión de la aplicación se podrían conseguir aplicaciones con mayor calidad.

Palabras clave

Realidad Aumntada, Puntos de interés, Máquinas de vector soporte, Andoir Studio.

Abstract

The objective of this Final Degree Thesis is to create a system of augmented reality. To this aim, we combine the tools and concepts of computer vision, Machine Learning and the use of graphics. Together, the shape an application that can detect an object and generate augmented reality on it.

After studying the different possibilities for the creation of applications of augmented reality, we have selected a technique without markers (Markerless). In order to fulfil our objectives, the design of the application has been arranged on a per stage basis:

First, the detection of points of interest for the detection and description of objects. Second, the training of a support vector machine. Third, the use of computer vision techniques. Fourth, the use of a graphic engine.

The model has been evaluated and the results are, in general, promising. However, we have found limitations when detecting the object when it's subjected to heavy rotations. To conclude, we believe that, in spite of limitations, the results of the developed applications are positive and motivate further research.

Keywords

Augmented Reality, Points of Interest, Support Vector Machines, Android Studio.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

En la actualidad se comienza a desarrollar gran cantidad de aplicaciones que se basan en Realidad Aumentada (RA, o AR por sus siglas en inglés). Empresas como Microsoft, Intel o Samsung están creando Hardware para utilizar la realidad aumentada en nuestro entorno diario, como vemos en la Figura 1.1

La realidad aumentada tiene muchas posibilidades para mejorar la interacción entre el usuario y el entorno. Además, la realidad aumentada no se limita única y exclusivamente a una mayor interacción visual con el entorno, sino que también se incluyen mejoras en otros sentidos como el auditivo o el olfativo.

En este trabajo, la realidad aumentada que vamos a proporcionar es sólo visual, incorporando aspectos visuales a nuestro entorno. Partiendo de un entorno con pocas posibilidades para interactuar, el poder incluir aspectos virtuales en dicho entorno lo convierte en una experiencia potencialmente atractiva. Entre estos aspectos virtuales, a los que también podemos referirnos como objetos virtuales añadidos, destacan la posibilidad de añadir información a un objeto real y la interactividad que puedes proporcionar a dicho objeto. Entendemos por objeto real cualquier objeto del entorno que detectamos a través de nuestra cámara y que es reconocido por un algoritmo que permite añadirle estos objetos virtuales. La detección de los objetos se realizará con la cámara de un *smartphone*.

La movilidad de los dispositivos móviles inteligentes ha sido la clave para potenciar esta tecnología. Esta situación motiva el diseño, la implementación y la evaluación de una aplicación para móvil, concretamente para dispositivos con sistema operativo Android, en la que buscará detectar un objeto multiplano y, a partir del cual, genera objetos en tres dimensiones que permitan en un futuro la interacción con el usuario.

Figura 1.1: Concepto de funcionamiento de las gafas de realidad aumentada Hololens, extraída de la página oficial de Microsoft.

1.2. Objetivos

El objetivo de este trabajo es la creación de una aplicación de realidad aumentada. Esto conlleva a crear un mecanismo para la detección de un objeto multiplano, tratándose en nuestro caso de un objeto básico, un cubo. Cuando el objeto sea detectado, procederemos a sacar su matriz de proyección. Con esta matriz podremos crear nuevos objetos virtuales en tres dimensiones alineados con el objeto físico.

Para cumplir nuestro objetivo final, hemos establecido los siguientes objetivos específicos:

1. Estudio del estado del arte

   Al comienzo del proyecto, se realizará un estudio para conocer cómo está formado un sistema de realidad aumentada. Las técnicas y algoritmo en el campo de visión artificial para la detección de objetos, y las herramientas software que se necesitarán para el desarrollo de la aplicación.

2. Detección de objeto

   Conociendo ya las técnicas necesarias se procederá al diseño y desarrollo de la aplicación en cuestión. En primer lugar se desarrollará la parte de detección del objeto físico con herramientas y técnicas de tratamiento de imágenes y aprendizaje automático.

3. Alineación de objetos virtuales y reales

   Se utilizarán las herramientas para el tratamiento de los elementos virtuales ajustándolo a la imagen que obtenemos de la cámara.

4. Evaluación de resultados

   Se evaluará el correcto funcionamiento de la aplicación.

1.3. Organización de la memoria

La memoria consta de los siguientes capítulos:

- Capítulo 1. Introducción.
- Capítulo 2. Estado del arte.
- Capítulo 3. Diseño y desarrollo.
- Capítulo 4. Evaluación.

- Capítulo 5. Conclusiones y trabajo futuro.
- Bibliografía.

Capítulo 2

Estado del arte

2.1. Introducción

Según Ronald Azuma [5], las bases de un sistema de realidad aumentada son tres:

- Combina elementos virtuales y reales en un entorno real.
- Ajusta la postura de los objetos virtuales con los reales.
- Se ejecuta en tiempo real, en tres dimensiones y es interactivo.

Estas bases siguen vigentes en el diseño de los sistemas de realidad aumentada, como se puede comprobar en otros trabajos como [1] y [6].

A partir de estos aspectos, podemos definir a un sistema de Realidad Aumentada como aquel que complementa al mundo real con objetos virtuales, de tal manera que los objetos virtuales se ajustan a la realidad. Estos sistemas que se puede ajustar dentro de la Figura 2.1.

A la hora de diseñar una aplicación de Realidad Aumentada debemos tener en cuenta diversos aspectos y problemas que surgen al mostrar la información en la

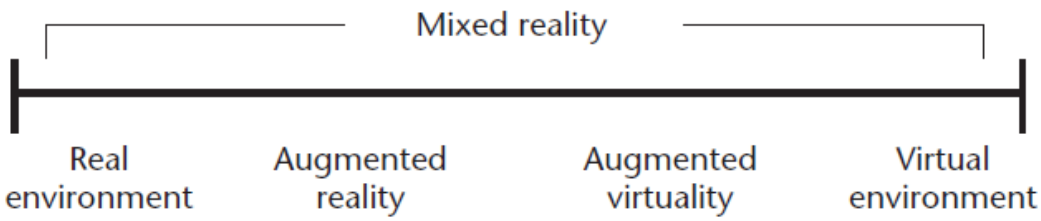

Figura 2.1: Entorno Real - Entorno Virtual. Fuente: [1]

pantalla del sistema de Realidad Aumentada. Así como factores humanos y problemas de percepción.

- **El problema del registro** o de estimación. Uno de los problemas básicos que limita las aplicaciones de realidad aumentada. Los objetos en el mundo real y el virtual tienen que estar perfectamente alineados, logrando que los dos mundos coexistan correctamente.

- **Oclusión de objetos reales.** Este problema va más allá de extraer información de profundidad de una escena; el sistema debe segmentar objetos individuales en ese entorno. En algunas situaciones, esta técnica permite la inserción de objetos virtuales y la supresión de objetos reales sin una reconstrucción 3D explicita del entorno.

- **Latencia.** Factor muy importante para que el sistema pueda funcionar correctamente y que el usuario no note un retardo en su funcionamiento.

Para la creación de aplicaciones de realidad aumentada encontramos diversos métodos que varían en cuanto a la posición de la pantalla, como en el método de generar los objetos virtuales. En este capitulo se hará un repaso de los tipos de Realidad Aumentada que encontramos actualmente y los métodos utilizados.

2.2. Realidad Aumentada

Podemos distinguir niveles de complejidad en las aplicaciones de realidad aumenta, Lens-Fitzgerald, cofundador de Layar[7], define 4 niveles según la tecnología empleada. En estos niveles profundizaremos en el apartado siguiente.

Otra manera de clasificar las aplicaciones de RA será el posicionamiento de la pantalla. Según [6] podemos clasificar las pantallas en las siguientes categorías:

- **Pantalla en la mano**. Gracias a los smartphone este tipo de pantallas se ha vuelto muy popular. Con la cámara del móvil captamos nuestro entorno y los nuevo elementos dentro de nuestro sistemas aparecen en la pantalla, y gracias al panel táctil podremos interactuar con ellos.

- **Pantalla en la cabeza**. Este tipo de pantalla comúnmente llamadas HMD (*head-mounted display*), este tipo de gafas debe permitir ver lo que ocurre a nuestro alrededor, y proyectar los objetos en el cristal de las mismas.

- **Pantalla proyectada**. Esta categoría de pantallas se basa en la proyección de las imágenes con la ayuda de un proyector. El objeto permanece inmóvil y el

proyector cambia la imagen según convenga. Aunque no hace falta que sea un objeto sólido, podemos ver un ejemplo de proyección en arena [8].

2.3. Niveles de complejidad

En esta sección se analiza los niveles de Realidad Aumentada mencionados anteriormente, que se distinguen en los métodos que utilizan.

Nivel 0 (Códigos QR)

Estas aplicaciones son las más básicas. Se basan en la detección de códigos QR para enlazar información, normalmente páginas web. Los códigos QR son matrices bidimensionales que registran la información en forma de pixeles blancos y negros que representan los bits. Como vemos en [9] la detección de estos códigos se basa en la detección de bordes. Primero se realiza un balanceo de la luz de la imagen de entrada. A continuación, con la imagen mejorada se pasa a escala de grises y se segmenta la imagen, gracias a esto se puede realizar una búsqueda de patrones del código QR.

Nivel 1 (Marcadores)

Los Marcadores o Marker-based son marcadores que se utilizarán como punto de partida para generar nuestra realidad aumentada [10]. La gran robustez y poco coste computacional hacen que esta técnica sea ideal para dispositivos móviles. Esta técnica se basa en la estimación de la rotación y posición de los marcadores.

Estimación de la rotación y posición También conocido como el problema de ***Perspectiva-n-point***, dado un conjunto de n puntos 3D y sus correspondientes proyecciones en 2D, junto con los parámetros de calibración de la cámara se tiene que calcular la rotación y traslación de la cámara respecto al mundo. Para resolver esto partimos del modelo de perspectiva de una cámara:

$$sp_c = K[R|T]p_w \tag{2.1}$$

Donde $p_w = [x\ y\ z\ 1]^T$es el punto homogéneo, $p_c = [u\ v\ 1]^T$es el punto de la imagen correspondiente, K es la matriz de los parámetros intrinsecos de la cámara, s es el factor de escala para el punto. Finalmente R y T son las rotación 3D y la

Figura 2.2: AR Markerless, resultado obtenido utilizando la mano para calcular la pose de la cámara. Extraído de [2].

traslación de la cámara. Desarrollando la ecuación:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_o \\ 0 & 0 & 1 \end{bmatrix}}^{K} \overbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}}^{R} \overbrace{\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}}^{T} \tag{2.2}$$

Donde f_xy f_y son la escala de la distancia focal, γ es el parámetro de sesgo que se asume que es cero, (u_0, v_0) es el punto principal y s el factor de escala de la imagen.

Nivel 2 (Markerless)

Esta técnica denominada *sin marcadores* (*Markerless*), se basa en generar RA sin ayuda de marcadores. En ella, se puede distinguir dos estrategias: el seguimiento basado en modelos y el movimiento de coincidencias [11]. Estas dos estrategias se basan en la correspondencia entre las características del mundo 3D y las características 2D de la imagen. En [2] crean un modelo basado en la mano, utilizando la segmentación de la imagen con el color de su mano y la posterior detección y seguimiento de los dedos. El resultado final puede verse en 2.2

Nivel 3 (Visión Aumentada)

Las tecnologías y aplicaciones en este nivel esta avanzado lentamente debido a su alta complejidad. Google empezó con sus Google Glass, pero sin mucha repercusión. Actualmente el que mas repuercusión tiene es Microsoft con sus HoloLens. El objetivo

Figura 2.3: HoloLens Microsoft utilizadas con el juego Minecraft. Imagen extraída de la página oficial de HoloLens.

de estas gafas es lograr la realidad mixta mezclando totalmente la virtualidad y la realidad.

2.4. Estrategias para el nivel 2

En este trabajo se presentará una aplicación de nivel 2, a continuación repasaremos algunos métodos que se utilizan en este nivel.

2.4.1. Puntos de interés

La localización de Puntos de Interés (PoI), es una técnica que se basa en la localización precisa y robusta de puntos invariantes a transformaciones típicas como traslaciones, rotaciones o cambios de escala.

En esta técnica de visión por ordenador se distinguen diferentes fases entre detección, descripción y correspondencia. La **detección** es un proceso que se encarga de buscar en la imagen los puntos más característicos, para lo que se utilizan detectores en el espacio escala que explicamos más adelante. La **descripción** se necesita para poder describir las regiones alrededor de los puntos localizados previamente. Una vez se han descrito estas regiones, se realiza un proceso de **correspondencia** entre dos imágenes.

2.4.1.1. Detectores de puntos de interés

Primera fase en el proceso de la localización de puntos de interés.

Detector Harris-Stephens

El detector Harris-Stephens[12], se presenta como una mejora del detector de esquinas de Moravec [13]. Esta mejora lo hace invariante a rotaciones e invariante a cambios de iluminación globales. También tiene algunos inconvenientes como la selección empírica del umbral de decisión y no es invariante a cambios de escala.

Espacio Escala y Representaciones piramidales

El espacio escala [14], es un tipo de representación multi-escala cuyo objetivo es encontrar estructuras estables en diferentes escalas y preservar el mismo muestreo espacial en todas las escalas. Esta representación se define mediante un conjunto de imágenes suavizadas aplicando un filtro sobre esta, normalmente se realiza con un filtro Gaussiano ya que este no produce nuevas estructuras en la imagen. Esta representación permite crear herramientas para la detección de puntos de interés en el espacio escala, logrando encontrar puntos robustos frente a la variación de escala en las imágenes.

Las representaciones piramidales[15] se presentan como un conjunto de versiones suavizadas de una señal, de tal manera que el número de píxeles disminuye en una versión a la siguiente por un factor constante. Esta técnica requiere poca memoria y su algoritmo es fácilmente paralelizable, pero no es invariante a traslaciones y aumenta la complejidad de mantener estructuras entre escalas.

Algoritmos

A continuación explicamos algunos algoritmos basados en el espacio escala o en las representaciones piramidales.

- **SIFT(Scale-Invariant Feature Transform)** Presentado por D.Lowe en 2004 [16], es una versión optimizada del detector Diferencias de Gaussianas, utilizando representaciones piramidales del espacio escala. Con esto logra reducir el número de suavizados y reduce su coste computacional.
- **SURF(Speeded-Up Robust Features)** Publicado como una versión más rápida de SIFT [17]. SURF utiliza cajas de filtro para su aproximación a Laplaciana de Gaussianas. La ventaja de aplicar el filtro de caja es su calculo, que es mucho menos costoso y aplicable en paralelo para diferentes escalas.

- **FAST (Features from accelerated segment test)** presentado en 2006 por Rosten[18], se presenta como una alternativa eficiente en la computación para la detección de esquinas, y se basa en el detector SUSAN.
- **ORB (Detector Oriented FAST and rotated BRIEF**), El detector ORB [4] presentado en 2011 junto con su descriptor como una alternativa eficiente a los detectores SIFT y SURF. Este detector se basa en FAST, creando representaciones piramidales mediante un filtro de Harris.

2.4.1.2. Descriptores de puntos de interés

En esta fase se procede a la descripción de los puntos para su posterior correspondencia.

- **BRIEF(Binary robust independent elementary features)**, Presentado en 2010 [19] , como una manera eficiente para la descripción de puntos de interés. Se puede usar con distintos detectores como SIFT o SURF y su uso se suele dar en aplicaciones en tiempo real dada su alta eficiencia.
- **ORB (Detector Oriented FAST and rotated BRIEF**), descriptor que se presenta junto con su detector que se ha nombrado antes. Este descriptor se basa en BRIEF añadiendo la característica de ser invariante a rotaciones.

2.4.1.3. Correspondencia entre imágenes

El objetivo es establecer correspondencia entre dos imágenes como un modelo y una escena. La correspondencia entre las imágenes se hace mediante los descriptores.

- Fuerza bruta: Este *matcher* se basa en encontrar la mejor correspondencia realizando un emparejamiento entre un descriptor del primer conjunto con todos los del segundo conjunto. Para ello utiliza el calculo de distancias entre los descriptores.
- FLANN (Scalable Nearest Neighbor Algorithms for High Dimensional Data), Biblioteca para realizar la correspondencia entre descriptores. Está formada por un conjunto de algoritmos optimizados para una búsqueda rápida de vecinos próximos en grandes conjuntos de datos.

2.4.2. Máquinas de vector de soporte

Las máquinas de vector soporte o *support vector machine* (SVM) propuesta por Vapnik [20] es una técnica de Aprendizaje Automático para problemas de clasifica-

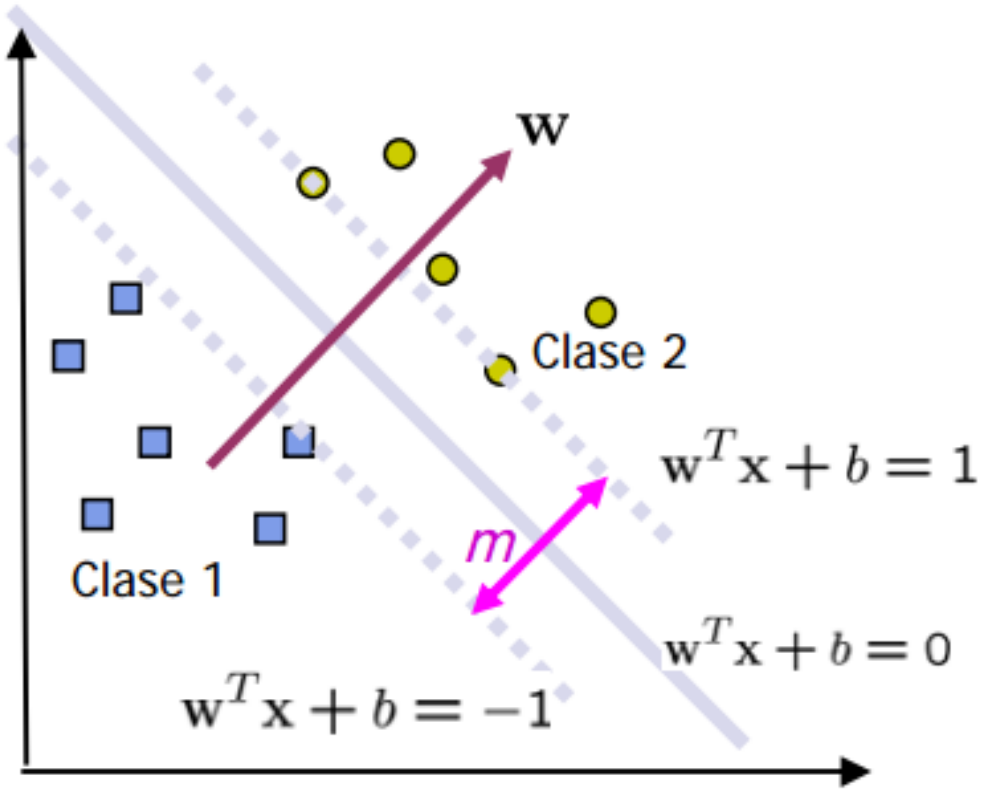

Figura 2.4: Separación de clases en las SVM. Gráfica extraída de [3]

ción. Su objetivo es encontrar formas de estimar dependencias funcionales a partir de un conjunto de datos de entrada. La técnica SVM transforma los datos a un espacio de dimensión superior a través de una función kernel, para poder encontrar un hiperplano que separe y maximice el margen m entre las clases como podemos ver en la Figura 2.4. La solución final del hiperplano podrá ser definida como vectores de soporte formados por la unión de puntos.

Para estos problemas de clasificación encontramos casos linealmente dependientes y los que no son linealmente dependientes.

2.4.2.1. Separación lineal

Dado un conjunto de datos iniciales S con dos clases. Los puntos $x_i \in \Re^N$ pertenecientes a cada clase definidas como $y_i \in -1, 1$ para cada punto i = 1, . . . , l.Donde $z = \varphi(x)$ es el vector en un espacio de características con un mapeo φ de $\Re^N$ a un espacio de características Z. Su objetivo es encontrar el hiperplano $w \cdot z + b = 0$ que separe las muestras en puntos de una clase. Podemos definir la siguiente función para clasificar los puntos x_i

$$f(x_i) = sign(w \cdot z_i + b) = \begin{cases} 1 & y_i = 1 \\ -1 & y_i = -1 \end{cases} \tag{2.3}$$

Donde $w \in Z$, $b \in \Re$ y $sign(\cdot)$ la función signo.

El conjunto de datos S será linealmente separable si existe un (w, b) tal que

$$\begin{cases} (w \cdot z_i + b) \geq 1 & y_i = 1 \\ (w \cdot z_i + b) \leq -1 & y_i = -1 \end{cases} \tag{2.4}$$

2.4.2.2. Separación no lineal

En estos casos necesitamos encontrar una función $K(\cdot, \cdot)$ que llamaremos kernel que transforma los datos a un espacio de dimensión superior para facilitar la separación lineal. La solución al problema del hiperplano queda definido como:

$$\begin{gathered} \text{mín} \left\{ \frac{1}{2} w \cdot w + C \sum_{i}^{l} \xi_i \right\} s.a \\ y_i(w \cdot z_i + b) \geq 1 - \xi_i, i = 1, \ldots, l \\ \xi_i \geq 0, i = 1, \ldots, l \end{gathered} \tag{2.5}$$

,donde el parámetro C es una constante y es definida empíricamente mediante la validación cruzada de los datos de entrenamiento.

Para poder trabajar con los datos que no son linealmente separables, se introducen las variables $\xi_i \geq 0$ de "holgura".

Finalmente la solución al hiperplano no lineal queda definida como

$$\begin{array}{c} Max\ W(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j K(x_i, x_{j)} \\ s.a\ \sum y_i \alpha_i = 0,\ 0 \leq \alpha_i \leq C,\ i = 1, \ldots, l \end{array} \tag{2.6}$$

donde α es un vector es un vector de multiplicadores de Lagrange positivos asociados con las constantes que no son separables linealmente.

La función de decisión del hiperplano:

$$f(x) = sign(w \cdot z + b) = sign(\sum \alpha_i y_i K(x_i, x_j) + b) \tag{2.7}$$

Para más información del funcionamiento de las SVM consultar [3].

2.5. Herramientas y tecnología utilizada

En la actualidad se encuentran disponibles una serie de herramientas para poder construir sistemas de realidad aumenta.

2.5.1. Visión Aumentada

Con el aumento de popularidad de las aplicaciones de realidad aumentada han surgido muchos librerias de visión aumentada que facilitan la creación de aplicaciones basadas en RA.

- **Vuforia**, SDK para el desarrollo de aplicaciones de realidad aumentada. Permite realizar tracking a objetos planos y a objetos en 3D. Es gratis su uso, pero hay que pagar licencia si se hace un uso comercial.

- **ARToolKit**, biblioteca para la creación de aplicaciones de realidad aumentada, Open Source. Es más limitada que Vuforia, solo permitiendo hacer tracking a planos.

- **OpenCV**, biblioteca Open Source más usada para el tratamiento de imágenes. La gran mayoría de los SDK de AR están basadas en esta librería. Está disponible en muchos lenguajes como C/C++, Python o Java.

- **LibSVM**, biblioteca Open Source de máquinas de vector de soporte, desarrollada por Chih-Chung Chang and Chih-Jen Lin.

2.5.2. Desarrollo para móviles

Para el desarrollo de aplicaciones Android existe el entorno de desarrollo oficial, Android Studio. También existe otras posibilidades, como Unity. Este motor esta ganando fuerza gracias a su gran compatibilidad de exportar para multiples plataformas.

2.5.3. Realidad Aumentada

Para crear elementos 3D para aplicaciones de realidad aumentada se utilizan motores gráficos. Existe un estándar llamado **OpenGL**, que es una API multilenguaje y multiplataforma para generar gráficos en 3D. También existen otras alternativas como el motor **Unity** que facilita el uso de gráficos.

2.6. Aplicaciones de la Realidad Aumentada

Dada la versatilidad que ofrece la Realidad Aumentada, se conocen aplicaciones en diversos campos. Entre los campos en los que la realidad aumentada aportando mayor valor.

En **educación** existen diversas herramientas, como LayAR o Aurasma, que permiten a los profesores crear contenidos con realidad aumentada fácilmente. Permite ofrecer gran cantidad de información, como textos, vídeos, etc; de forma que puede añadir valor a la exploración del mundo real [21].

En **videojuegos** con el uso de GPS se aumentan las posibilidades de interacción con el mundo real, lo que consigue ofrecer una experiencia de juego muy completa. Un claro ejemplo lo tenemos en el conocido juego Pokemon Go.

También podemos encontrar sistemas completos como Project Tango de Google. El objetivo de este proyecto es conseguir una herramienta de visión artificial que permita registrar en 3D el entorno que nos rodea de forma precisa, y aprovechar las oportunidades que nos puede ofrecer el mapear nuestro entorno.

2.7. Conclusiones

En este capítulo hemos hecho una revisión del estado del arte de la realidad aumentada. Primero se establecen las bases de la realidad aumentada y los factores que se deben tener en cuenta en el diseño de una aplicación.

En segundo lugar, se repasan los niveles que separan las distintas aplicaciones de realidad aumentada. Destacando los del nivel 2 como los puntos de interés. Tam-

bién se describe la técnica SVM, que permite un aprendizaje automático basado en descriptores, para una mejora de la clasificación de imágenes.

Por último, se exponen las características técnicas de algunas herramientas del mercado que facilitan la creación de contenidos con realidad aumentada y se recogen las aplicaciones más relevantes del uso de esta técnica en el día a día de algunos sectores.

Después de haber repasado las bases que fundamentan este trabajo, se concluye este capítulo con la definición del sistema desarrollado. Nuestro sistema se sitúa en el nivel 2 y utiliza las máquinas de vector de soporte entrenadas con los descriptores ORB de un objeto. Con el entrenamiento de nuestro sistema de machine learning lograremos la detección de un objeto en la escena que captura nuestro teléfono móvil para más adelante poder generar objetos 3D mediante OpenGL.

Capítulo 3

Diseño y desarrollo

3.1. Introducción

En este capitulo se describe las técnicas y herramientas empleadas para desarrollar un sistema de realidad aumenta. El objetivo de este sistema será encontrar un objeto multiplano, cuyas caras han sido utilizadas para entrenar previamente una máquina de vector de soporte. Una vez detectado nuestro objeto en las imágenes de entrada, el sistema procederá a calcular la matriz de rotación y traslación para poder generar los modelos 3D.

El funcionamiento de el sistema propuesto lo vemos en la Figura 3.1. En la figura se distinguen las partes de Entrenamiento y Aplicación. La primera se centra en el entrenamiento de las máquinas vector de soporte. La parte de Aplicación que será la encargada cargar los datos que hemos obtenido para generar la RA.

3.2. Caracterización

Detección

Dado un conjunto de imágenes de entrada($I_1 ... I_n$), se procede a la detección de los puntos de interés mediante el detector ORB, por su rapidez y robustez frente a rotaciones[4].

ORB propone al detector FAST robustez frente a cambios de orientación en los PoIs. Para ello ORB utiliza la intensidad de los centroides, asumiendo que la intensidad alrededor de un punto es proporcional al desplazamiento desde el centro y se puede utilizar como orientación. Gracias al uso de esta técnica se obtiene buenos resultados cuando se quiere recuperar la rotación inicial.

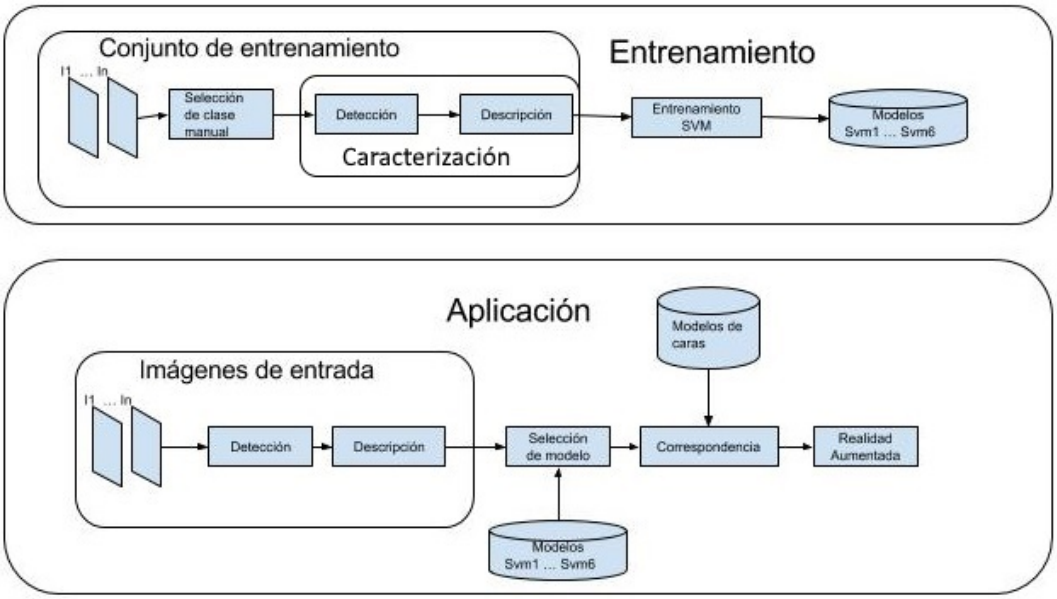

Figura 3.1: Diagrama del sistema propuesto.

Descripción

Para la descripción hemos usado el descriptor de ORB, que está basado en BRIEF. BRIEF es un descriptor eficiente pero pobre ante rotaciones de puntos de interés. Para permitir que BRIEF sea invariante a rotaciones ORB propone un método que utiliza la orientación de los puntos de interés. Para cualquier conjunto de características de n pruebas binarias en la posición de los puntos de interés (x_i, y_i), se define la matriz

$$S = \begin{pmatrix} x_1, \ldots, x_n \\ y_1, \ldots, y_n \end{pmatrix}$$

Utilizando la orientación θ y la matriz de rotación $\mathbf{R}_\theta$se construye$\mathbf{S}_\theta$(versión dirigida) de $\mathbf{S}$:

$$\mathbf{S}_\theta = \mathbf{R}_\theta \mathbf{S}$$

Discretizando el angulo con incrementos de $2\pi/30$, se construye una tabla previamente procesada de búsqueda de patrones BRIEF. Mediante test binarios buscan aquellas que tengan una alta varianza y que no estén correlacionadas. Con estos resultados y un algoritmo de aprendizaje automático obtienen lo que llaman rBRIEF. En la Figura 3.2, extraída del articulo original, se compara ORB con los descriptores de SIFT, SURF y BRIEF. Estos resultados se dan en términos de coincidencia correcta con un conjunto de pruebas con ruido Gaussiano de desviación típica 10 y media

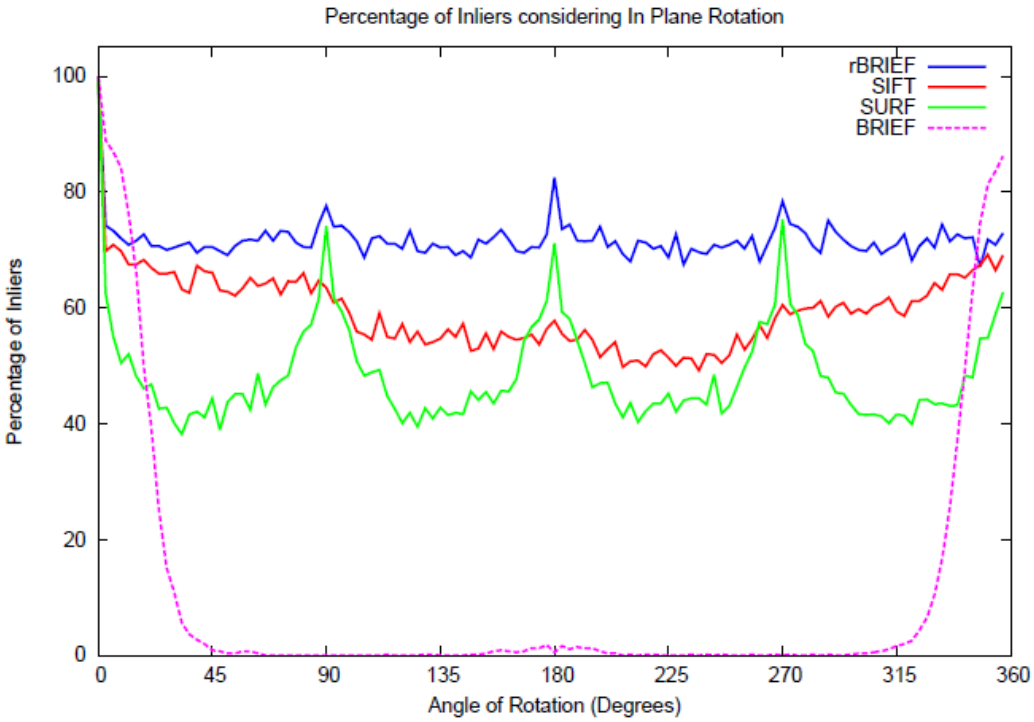

Figura 3.2: Rendimiento de matching de SIFT, SURF, BRIEF con ORB(oFAST + rBRIEF) utilizando rotaciones sintéticas. Adaptada de [4]

nula.

Se aprecia que los resultados de BRIEF caen cuando se pasa de los 10 grados. Obteniendo ORB los mejores resultados frente a SIFT y SURF.

3.3. Generación de modelos

Para el entrenamiento de las máquinas, se ha considerado dos estrategias. La creación de una máquina multiclase con seis clases o seis maquinas monoclases.

Las SVM seleccionadas son del tipo C [20], con función de decisión:

$$sgn(w^T\varphi(x) + b) = sgn(\sum y_i\alpha_i K(x_i, x) + b) \tag{3.1}$$

, problema de optimización:

$$min\ \frac{1}{2}w^T w + C\sum \xi_i \tag{3.2}$$

y el kernel radial Gaussiano (RBF)

$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2} \tag{3.3}$$

Los parámetros (C, γ) son variables que tenemos que calcular mediante la validación cruzada, que separa los datos de entrenamiento en pequeños grupos y calcula estos valores.

El objetivo de estas máquinas será el filtrado de los descriptores que no pertenezcan al objeto que estamos buscando.

3.4. Correspondencia entre modelos

Para la correspondencia entre descriptores de la escena y nuestro modelos de caras, utilizamos el Brute-Force Matcher. Con los descriptores de la escena filtrados por las SVM se comparan con el conjunto de descriptores de los modelos, de esta manera obtenemos la correspondencia entre los puntos. Para hacer el emparejamiento Brute-Force utiliza la distancia de Hamming. La distancia de Hamming $d(x, y)$ entre dos vectores $x, y \in F^{(n)}$es el numero de posiciones del vector que son diferentes. Es decir:

$$F_2^{(5)}\ d(00111, 11001) = 4$$

3.5. Generación de Realidad Aumentada

El último bloque es el que se encarga de generar los objetos 3D a partir del plano detectado, es decir, una de las caras de nuestro objeto en la imagen. Se calcula el campo de visión de la lente, que obtenemos previamente de los parámetros de la cámara, y calculamos la proyección para el motor gráfico. La proyección en el motor gráfico está basado en seis planos o Normalized Device Coordinates (NDC) diferente al de las cámara que esta basado en el modelo Pinhole(ver Figura 3.3).

Realizamos un emparejamiento entre los descriptores del modelo de la clase ganadora y para localizar el plano de una de las caras de nuestro objeto. Cuando tenemos a los candidatos finales resolvemos el problema *Perspective-n-Point* con los puntos del modelo y la escena, el resultado de esta operación no proporciona la pose de nuestro objetivo. Finalmente con estos datos el motor gráfico en capaz de representar objetos 3D ajustando su posición y rotación al plano detectado.

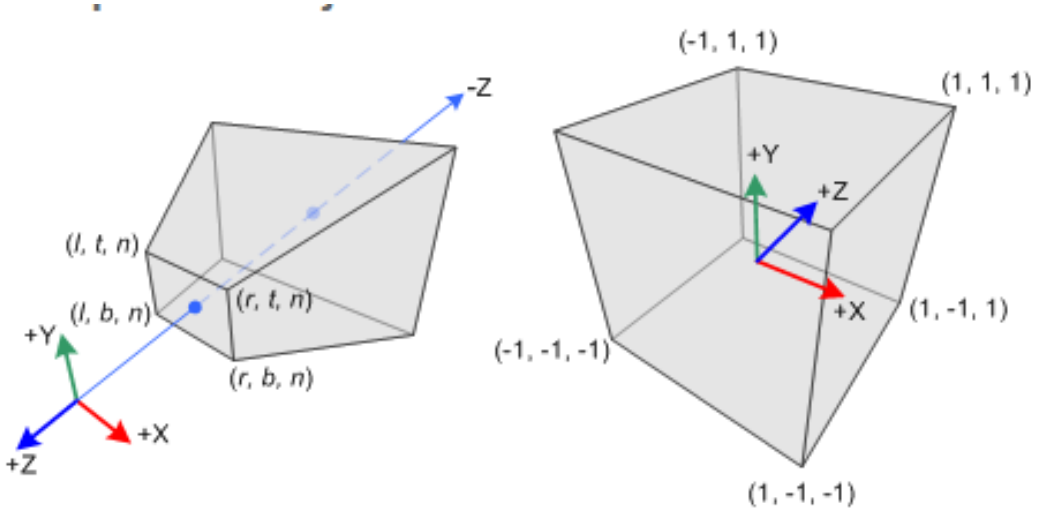

Figura 3.3: Comparación de la proyección pin-hole y la intersección de seis planos que utiliza OpenGL. Gráfica obtenida de http://www.songho.ca

3.6. Aplicación diseñada

En esta sección explicamos con detalle los bloques que se ven en la Figura 3.1.

Para implementar la aplicación se ha investigado las posibles maneras de incorporar una librería de procesado de imagen a Android, también se ha investigado el funcionamiento de motores gráficos para móviles y librerías para implementar máquinas de vector de soporte.

Finalmente se ha decidido utilizar la librería **OpenCV**[22] para el análisis de imágenes, y **LibSVM**[23] para la creación de las máquinas de vector de soporte, esta librería facilita la validación cruzada. Estas librerías correrán tanto sobre aplicaciones de escritorio como en la aplicación final de móvil. Se decidió utilizar **OpenCV** ya que posee una API propia en Java, que es el lenguaje principal utilizado por todo el sistema y compatible con aplicaciones Android y su gran documentación en la red. Finalmente para producir los gráficos 3D se utiliza la librería **OpenGL**, que un estándar libre para aplicaciones que utilizan gráficos. También se utilizarán scripts en Matlab para crear las mascaras que nos ayudarán a filtrar del descriptores. En esta sección se especifica los programas implementados del diseño presentado. La aplicación Android se desarrolla con **Android Studio**, el entorno de desarrollo oficial de Android.

3.6.1. Entrenamiento

Selección de clase manual

En este bloque obtenemos las mascaras de cada imagen para poder filtrar los descriptores de cada cara. Para ello ejecutamos el script en Matlab "anotation.m" en este script con la ayuda de la función "fit_ellipse.m" seleccionamos cinco pixeles de la imagen y calcula la mascara. Las imágenes se muestran en la Sección 4.2.

Detección y Descripción

Esta parte se encarga de la obtención de todos los descriptores para el futuro entrenamiento de las SVM. Para ello tenemos el programa en Java, "MainClasificador.java", este programa carga todas las mascaras y todas las imágenes y crea un fichero con todos los descriptores. Cuando tenemos todos los descriptores los escalamos para su entrenamiento. Escalamos los datos para evitar que los atributos con rangos altos dominen sobre los atributos de rangos pequeños. Otra ventaja es evitar dificultades numéricas durante el cálculo del kernel. Para escalar los datos ejecutamos el programa en Java "ScaleSvm.java" del paquete Libsvm,este programa escala los datos en el rango [-1,+1]. Con los datos escalados separamos los descriptores entre datos de test y los datos de train y los desordenamos de manera aleatoria para su entrenamiento.

Entrenamiento SVM

Con los datos separados pasamos a el entrenamiento de las SVM. Para el núcleo se ha utilizado RBF, que es el recomendado por la documentación de LibSvm para un entrenamiento inicial. Hay dos parámetros (C, γ) que no sabemos de antemano cuales son sus mejores valores. Para ello utilizaremos la validación cruzada, con el programa en Python, "grid.py", con n-flod con valor 3. Esto separa los datos de entrenamiento en tres partes para la validación de los datos de entrenamiento. Cuando tenemos los valores de (C, γ) de la validación cruzada pasamos a la generación de los modelos que posteriormente utilizará la aplicación Android para poder predecir que cara del cubo se esta viendo en la escena. Para la generación de los modelos ejecutamos el programa en Java "TrainjLibSvm.java", este programa entrena a los modelos con los valores obtenidos de la validación cruzada. Los resultados de la validación cruzada se ven el la sección 4.3.

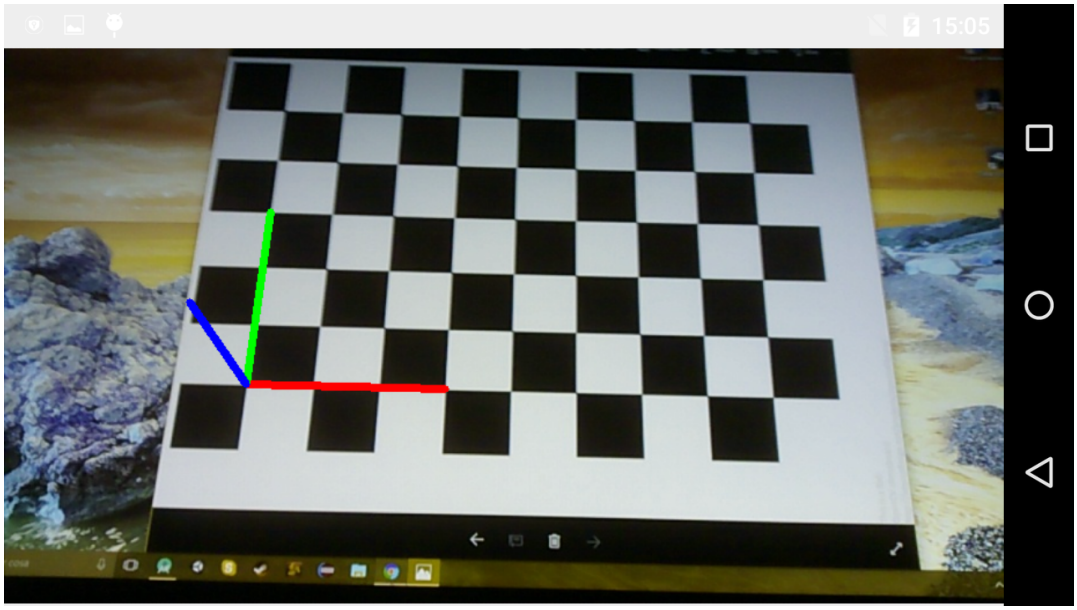

Figura 3.4: Resultado que obtenemos cuando ejecutamos FindChessActivity.java.

3.6.2. Aplicación Android

Calibración de la camara

En primer lugar procedemos a la calibración de la cámara del móvil. OpenCV te facilita la aplicación para ello. Con los datos de la cámara comprobamos si estos valores eran los correctos. Para ello tenemos la actividad "FindChessActivity.java" que con los datos de la cámara calcula la pose de un tablero de ajedrez. En la Figura 3.4 se muestra la salida.

Actividad principapal

El flujo de esta actividad la podemos ver en la Figura 3.5.

La actividad principal implementa la interfaz de Opencv "implements CvCameraViewListener2" con esta interfaz recibimos los fotogramas que capta la cámara. Al iniciar la aplicación calcula los descriptores de los seis modelos y prepara la vista del motor de gráficos .

La actividad principal llamará al método "onCameraFrame()" cuando la cámara esté activa, este recibe como parámetro de entrada el frame que capta la cámara y devuelve lo que queremos que se muestre.Cuando recibimos el frame llamamos a la función "findPose()" de la clase "ComputeFrame.java". Este método sera el encargado de analizar la imagen de entrada, filtrando los descriptores con las SVM. También

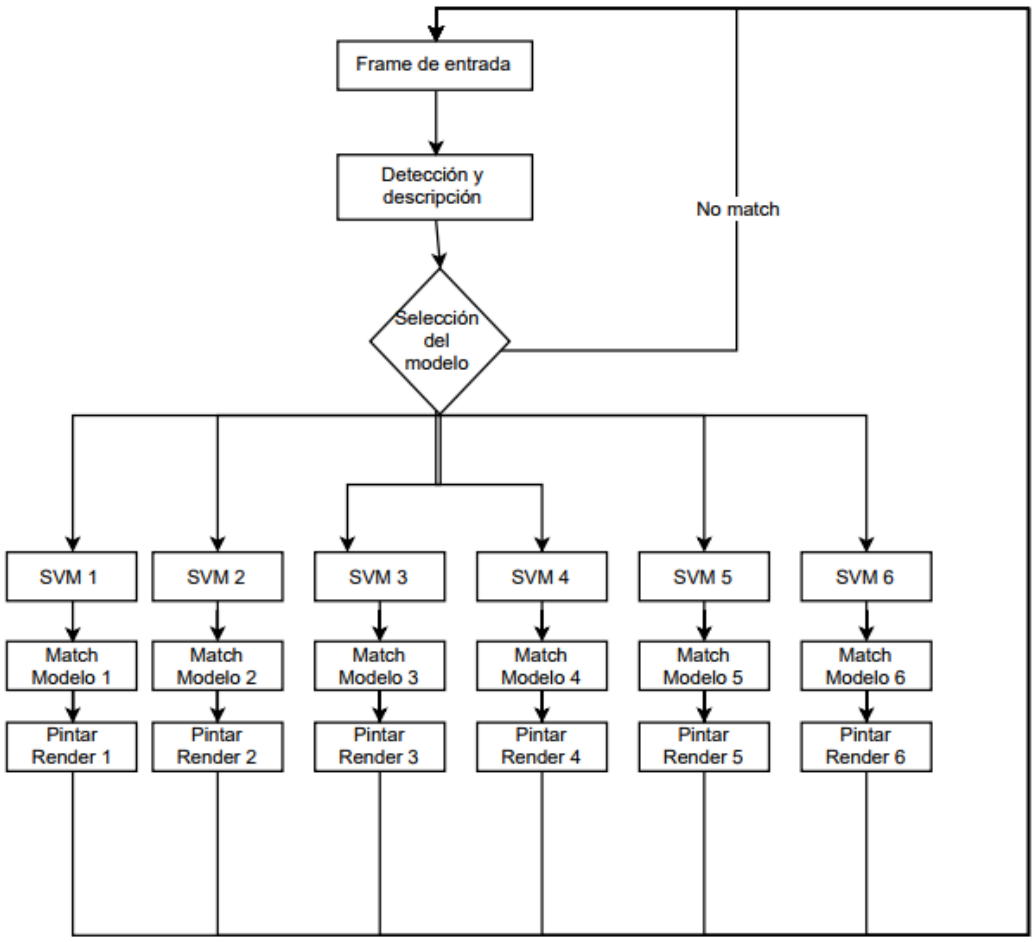

Figura 3.5: Diagrama de flujo aplicación.

```
//Ajustamos la vista con las cordenadas de la pantalla
final int adjustedWidth = (int)(mSurfaceHeight *
        cameraProjectionAdapter.getAspectRatio());
final int marginX = (mSurfaceWidth - adjustedWidth) / 2;
gl.glViewport(marginX, 0, adjustedWidth, mSurfaceHeight);

//Definimos la matriz de proyección
gl.glMatrixMode(GL10.GL_PROJECTION);
float[] projection = cameraProjectionAdapter.getProjectionGL();
gl.glLoadMatrixf(projection, 0);

//Definimos la posición del objeto
gl.glMatrixMode(GL10.GL_MODELVIEW);
gl.glLoadMatrixf(pose, 0);
```

Figura 3.6: Ejemplo de alineación de un objeto 3D con el real.

está "findPoseDistances()" que analiza la imagen pero sin usar las SVM.

Con la imagen de entrada primero sacamos los puntos de interés ORB y a continuación sus descriptores. Cuando tenemos el conjunto de descriptores se los pasamos a las SVM. La clase con más votos positivos será la que se utilizará para realizar el match. Se seleccionan los mejores y calculamos la homografía entre los puntos del modelo y la escena. Con la homografía calculada, transformamos las cuatro esquinas del modelo con la función de OpenCV "Core.perspectiveTransform()" a la pose de la escena, descartando los resultados no convexos. Ahora con las referencias en 3D, 2D y con la matriz de la cámara se calcula los vectores de rotación y traslación con "Calib3d.solvePnP()" de OpenCV. Con estos datos finalmente pintamos el render en 3D. En la Figura 3.6 vemos como se alinea los objetos 3D con los reales.

Capítulo 4

Evaluación

4.1. Introducción

En este capitulo se explica como se ha generado la base de datos de descriptores a partir de fotos de un objeto y se expondrán los resultados de la evaluación de las SVM y su posterior incorporación de la aplicación en Android.

4.2. Generación del Corpus para entrenamiento y validación

Para la generación de la base de datos para el entrenamiento, separamos las fotos en una, dos o tres caras visibles. Por cada imagen de una o varias caras se realizaron nueve fotos. Tres variando la escala de nuestro objetivo, el cubo. Tres variando las condiciones de luz y otras tres variando los ángulos de visión. En total obtuvimos 374 imágenes de nuestro modelo, con estas imágenes se generan a mano las mascaras de cada una para poder distinguir entre los descriptores de cada clase. Finalmente con la caracterización se ha obtenido un total de 157958 descriptores.

	Esc 1	Esc 2	Esc 3	Luz 1	Luz 2	Luz 3	Rot 1	Rot 2	Rot 3
Clase 1	1	1	1	1	1	1	1	1	1
Clase 2	1	1	1	1	1	1	1	1	1
...									
Clase 1y 2	1	1	1	1	1	1	1	1	1
...									
Clase 1,2 y 3	1	1	1	1	1	1	1	1	1

Tabla 4.1: Tabla utilizada para la creación del corpus.

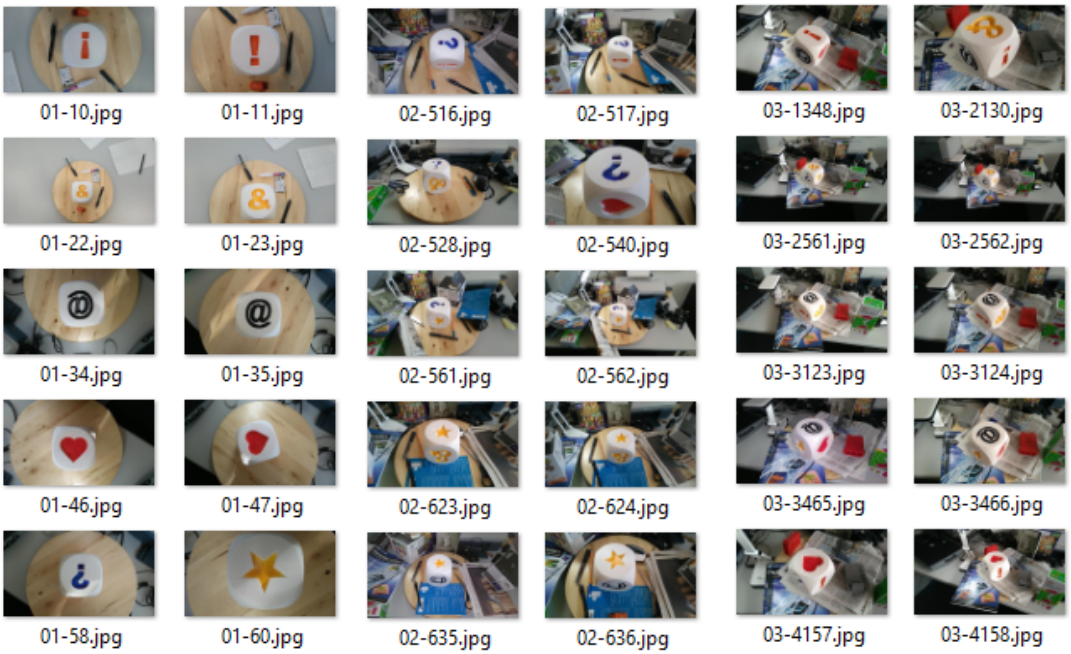

Figura 4.1: Pequeña muestra de las imágenes capturadas.

	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6
Clase 1	0,438	0,089	0,139	0,086	0,203	0,045
Clase 2	0,065	0,429	0,227	0,057	0,157	0,063
Clase 3	0,044	0,092	0,670	0,036	0,124	0,031
Clase 4	0,078	0,081	0,115	0,533	0,153	0,036
Clase 5	0,098	0,096	0,189	0,070	0,501	0,044
Clase 6	0,063	0,135	0,160	0,057	0,131	0,45

Tabla 4.2: Matriz de confusión de los datos de entrenamiento.

En la Figura 4.1 se muestra una pequeña parte de esta base de datos.

4.3. Evaluación de la separabilidad de los puntos de interés

Cuando tenemos todos los descriptores del objeto generados, creamos una matriz de confusión para la validación del conjunto de entrenamiento. Con el script de Matlab "confusion_media.m" que utiliza la función "knnsearch()". Esta matriz muestra los porcentajes de de aciertos de cada clase.Las filas representa el porcentaje total de muestras, mientras las columnas las predicciones de cada clase.

Los resultados de la Tabla, se pueden considerar aceptables. Se ve como el por-

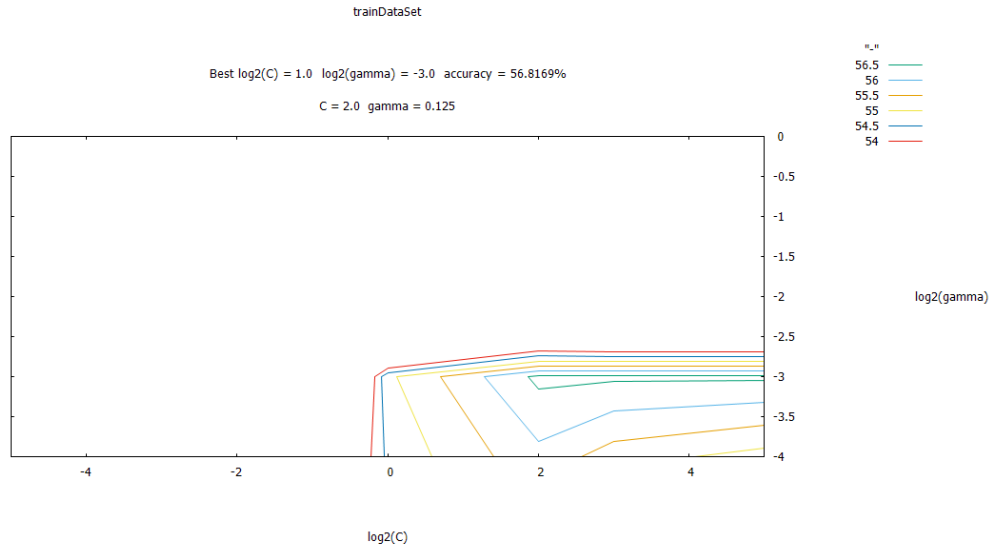

Figura 4.2: Gráfico con los resultados de la validación cruzada de la SVM con varias clases.

centaje de la diagonal principal es el mayor de cada fila, indicando un grado de separabilidad entre las clase. Se puede apreciar también como las clases 3 y 5 son las que más falsos positivos generar a las demás clases.

4.4. Configuración de las máquinas de vector soporte

Para el entrenamiento de las SVM se contemplan dos estrategias, una SVM multiclase o seis SVM monoclase. Los resultados de la validación cruzada de la SVM multiclase los vemos en la Figura 4.2. Los resultados de la otra estrategia se muestran en las Figuras 4.3, 4.4 y 4.5.

El mejor resultado para la maquina multiclase es de 56.5 % de aciertos.

Los resultados de las seis SVM son los siguientes, la clase 1 obtuvo un porcentaje de acierto de 91.9 %, la clase 2 un 86.38 %, la clase 3 un 79.05, la clase un 94.02 %, la clase 5 un 81.72 y la clase 6 un 94.06.

Con estos resultados se ha seleccionado la estrategia de seis SVM monoclases, ya que los resultados de su validación cruzada son significativamente más alto en todas las clases.

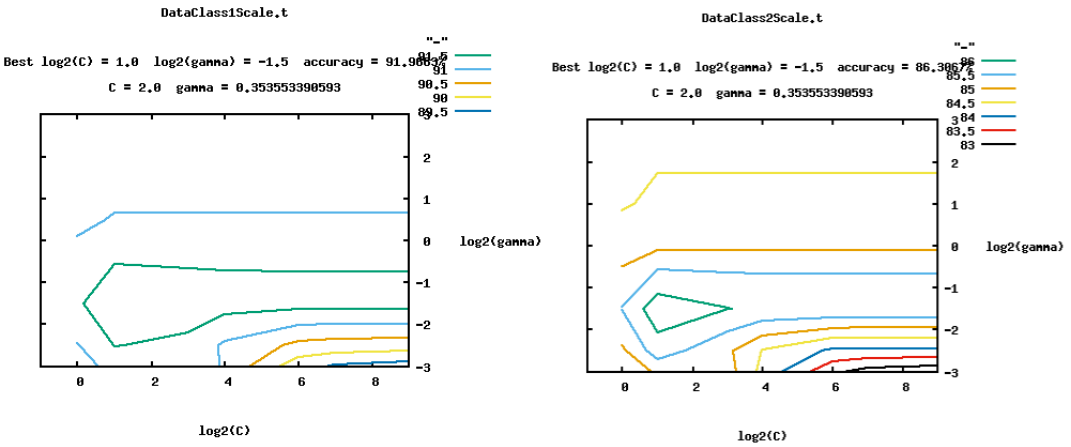

Figura 4.3: Gráfico con los resultados de la validación cruzada de la SVM 1 y 2.

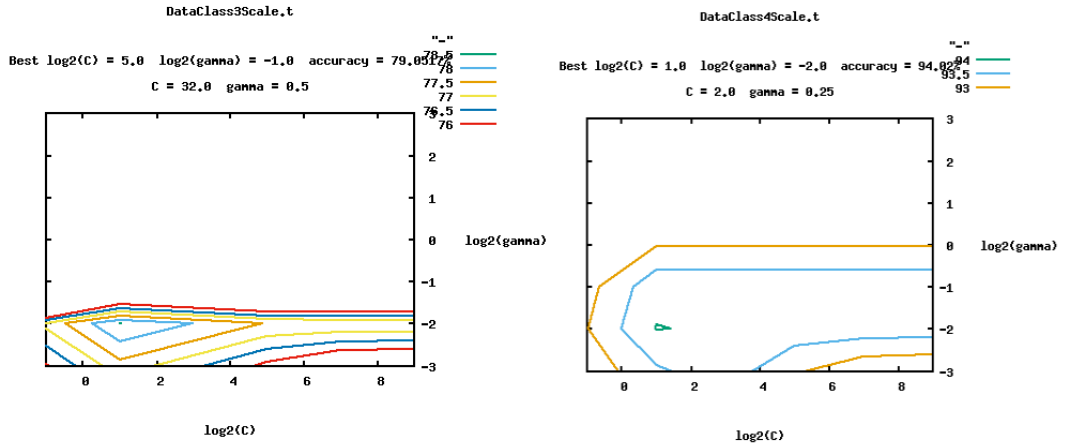

Figura 4.4: Gráfico con los resultados de la validación cruzada de la SVM 3 y 4.

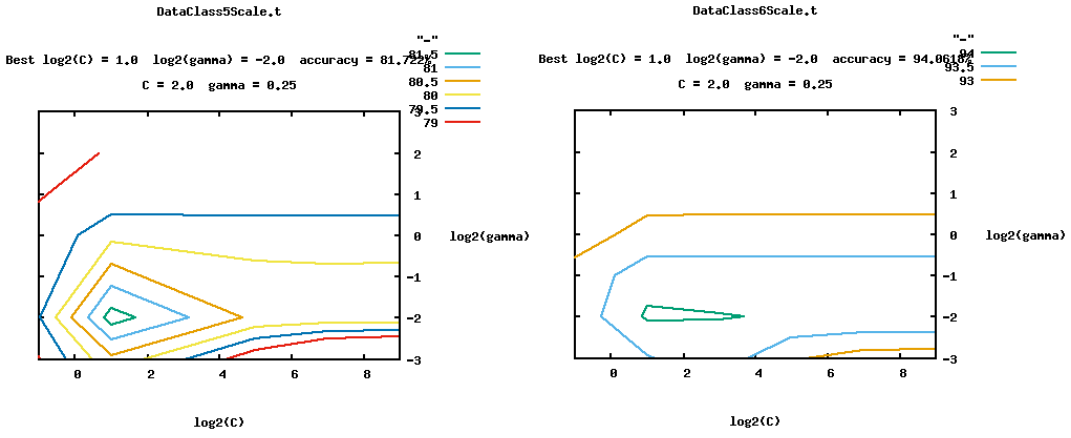

Figura 4.5: Gráfico con los resultados de la validación cruzada de la SVM 5 y 6.

4.5. Validación de modelos

Con los datos de test se evalúa la precisión, y la sensibilidad de las seis SVM. También se realiza barrido para encontrar un umbral que maximice estos resultados. Para ello se ha implementado "TestJLibsvm.java" que carga todos los descriptores de test de cada máquina y calcula la precision y sensibilidad de nuestro sistema.

Siendo la precisión

$$P = \frac{TP}{TP + FP}$$

y la sensibilidad o recall

$$R = \frac{TP}{TP + FN}$$

donde TP (verdadero positivo), es el número de aciertos correctos, FN(falsos negativos) a aquellas apariciones de descriptores de la clase y el sistema a dado por falsas. FP, falso positivo, son aquellos que el sistema predice que son de nuestra clase y no lo son.

Estos resultados se muestran en la Figura 4.6. Debido a sus diferencias en los resultados las gráficas tienen diferentes valores en los ejes de coordenadas.

Las gráficas muestran como todas las máquinas tienes una baja sensibilidad o recall, excepto la máquina 3 que se logra tener unos resultados aceptables. La baja sensibilidad implica que nuestro sistema le cuesta predecir correctamente la cara del objeto. Por otro lado la precisión de todas las máquina nos indica que distinguen correctamente los descriptores que no son de su clase.

4.6. Evaluación del sistema

En este apartado comentaremos los resultado que se han obtenido al incorporar los modelos a la aplicación móvil y crear realidad aumentada.

4.6.1. Visualización de la Realidad Aumentada

Al incorporar las seis SVM la aplicación no se han obtenido los resultados deseados. Cargar todos los modelos en la aplicación limita mucho su uso. Se ha realizado pruebas con una sola máquina debido a su alto coste. Se han logrado buenos resultados en el ajusto de los render 3D sobre el objeto real, pero el rendimiento es demasiado bajo.

En la Figura 4.7 se muestra una secuencia de imágenes como la aplicación ajusta la escala del render 3D a la escala del objeto detectado.

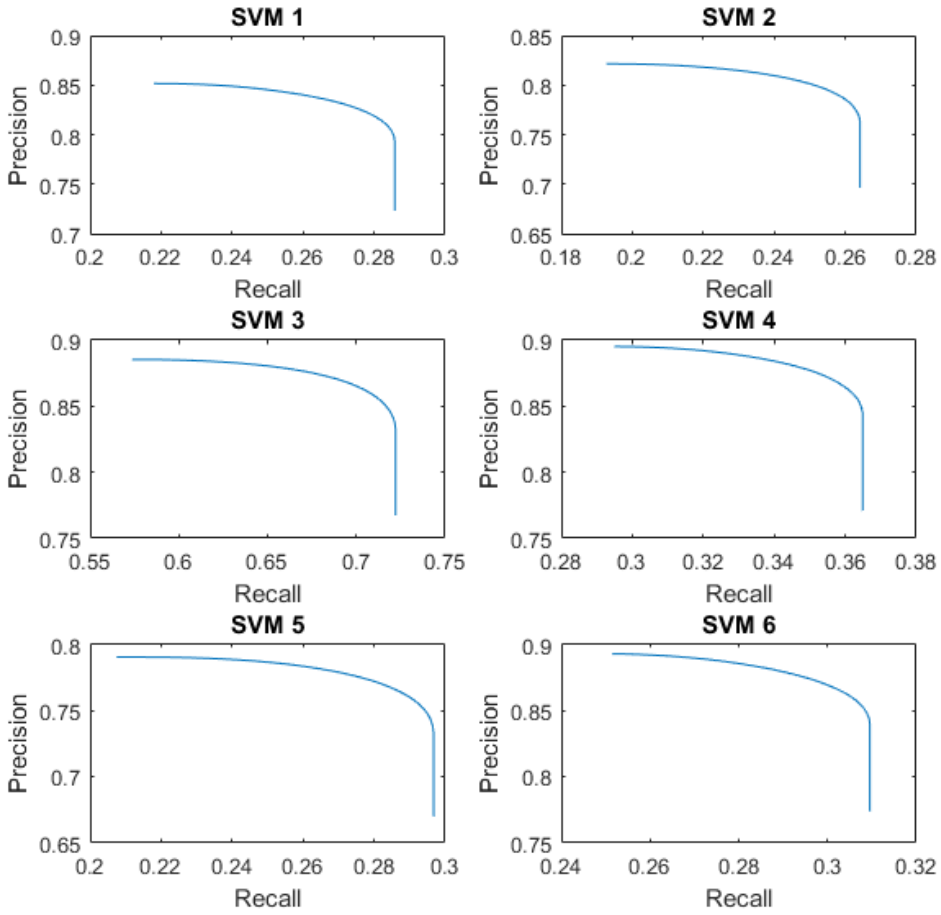

Figura 4.6: Resultados de test de las SVM. Se muestra los valores obtenidos con el barrido de un umbral para maximizar los resultados.

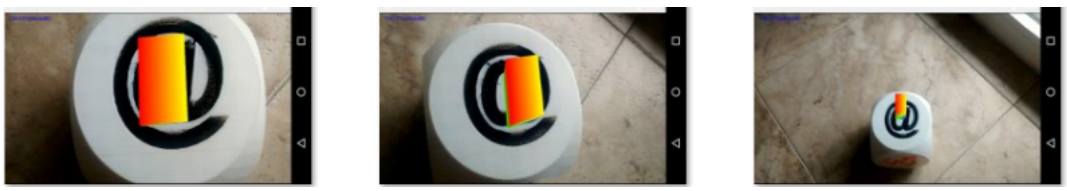

Figura 4.7: Resultados en diferentes escalas del objeto.

4.6.2. Discusión: Ventajas y Limitaciones

El sistema se ve limitado por la gran cantidad de datos que tiene que analizar y comparar con todas las SVM. Los datos positivos del sistema es que logra hacer una alineación correcta de los elementos generados por el motor gráfico y el objeto detectado con la cámara.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

Para lograr los objetivo propuestos en la 1.2, se han investigado en profundidad las técnicas que se utilizan en la creación de aplicaciones de realidad aumentada. De cara a la detección de objetos, el estudio de estas técnicas implica conocer, por una parte, las técnicas de visión por ordenador y, por otra parte, las técnicas de aprendizaje automático. De cara a la generación de gráficos, se han estudiado herramientas para la creación de estos a partir del objeto detectado.

Para la detección de objetos, se ha logrado crear un sistema para la detección de un objeto mediante técnicas de visión por ordenador y aprendizaje automático, con la detección del objeto se ha hecho con puntos de interés y se ha entrenado una máquina de vector de soporte con los descriptores de dichos puntos. Para ello, se ha desarrollo un sistema basado en Java que utiliza OpenCV para la visión por ordenador, para la carga y tratamiento de una base de datos de imágenes, y con los resultados de estas imágenes se ha entrenado una máquina de vector de soporte con LibSVM.

Para la generación de gráficos, se ha investigado las técnicas para poder ajustar elementos detectados con la cámara y elementos generados con un motor gráfico. Para ello, se han utilizado técnicas de visión por ordenador con OpenCV que se ajustan al motor gráfico. Con los valores resultantes de esta técnica se han configurado los gráficos, utilizando la librería OpenGL.

Para el entrenamiento de las máquinas de vector soporte, se ha creado una base de datos con imágenes de nuestro objeto. Para ello se ha capturado las diferentes caras de nuestro objetivo a detectar variando el ángulo, escala y luminosidad.

Para finalizar se ha evaluado la aplicación, se ha comprobado que el sistema funciona pero que debido al alto coste la aplicación no actuá con el rendimiento deseado.

Se han experimentado diversos problemas para su funcionamiento en Android, lo que sin duda ha supuesto una dificultad para la realización de este trabajo.

5.2. Trabajo futuro

A la vista de los resultados que se han obtenido en este trabajo se propone trabajar en:

- Mejorar el rendimiento de toda la aplicación, por su bajo rendimiento para trabajar en tiempo real en Android. Para esto tendríamos varias opciones. Una de estas opciones sería utilizar la librería en C++ de OpenCV, que podría aumentar el rendimiento en Android. Esta opción presentaba dos limitaciones, por una parte, este método se encuentra en pruebas, por lo que no se recomienda su uso; por otra parte, aumentaría la complejidad del diseño y desarrollo de la aplicación. Otra de las opciones sería utilizar unas librerías diferentes a OpenCV para la detección del objeto, como por ejemplo Vuforia.
- Para mejorar la creación de gráficos 3D, se podría utilizar un motor como Unreal o Unity. Con estos motores, la gestión de los modelos 3D facilitaría el desarrollo de la aplicación. Estos motores facilitan la carga de modelos 3D y la interacción entre ellos, al incorporar herramientas para trabajar con los gráficos a más alto nivel. Con esto podríamos superar las limitaciones que se presentan actualmente al trabajar a bajo nivel con OpenGL, de forma que mejoraría la interacción entre los modelos 3D, ya que actualmente se trabaja con una interacción básica. De esta manera la aplicación podríamos desarrollar aplicaciones más complejas.

Bibliografía

[1] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *Computer Graphics and Applications, IEEE*, vol. 21, no. 6, pp. 34–47, 2001. XI, 5

[2] T. Lee and T. Hollerer, "Handy ar: Markerless inspection of augmented reality objects using fingertip tracking," in *Wearable Computers, 2007 11th IEEE International Symposium on*, pp. 83–90, IEEE, 2007. XI, 8

[3] G. A. Betancourt, "Las maquinas de soporte vectorial (svms)," *Scientia et technica*, vol. 1, no. 27, 2005. XI, 12, 14

[4] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2564–2571, IEEE, 2011. XI, 11, 17, 19

[5] R. T. Azuma, "A survey of augmented reality," *Presence: Teleoperators and virtual environments*, vol. 6, no. 4, pp. 355–385, 1997. 5

[6] D. Van Krevelen and R. Poelman, "A survey of augmented reality technologies, applications and limitations," *International Journal of Virtual Reality*, vol. 9, no. 2, p. 1, 2010. 5, 6

[7] C. P. Espinosa, "Realidad aumentada y educacion: analisis de experiencias practicas," *Pixel-Bit. Revista de Medios y Educación*, no. 46, pp. 187–203, 2015. 6

[8] H. Ishii, C. Ratti, B. Piper, Y. Wang, A. Biderman, and E. Ben-Joseph, "Bringing clay and sand into digital designâcontinuous tangible user interfaces," *BT technology journal*, vol. 22, no. 4, pp. 287–299, 2004. 7

[9] Y.-H. Chang, C.-H. Chu, and M.-S. Chen, "A general scheme for extracting qr code from a non-uniform background in camera phones and applications," in *Multimedia, 2007. ISM 2007. Ninth IEEE International Symposium on*, pp. 123–130, IEEE, 2007. 7

[10] D. Wagner, T. Langlotz, and D. Schmalstieg, "Robust and unobtrusive marker tracking on mobile phones," in *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on*, pp. 121–124, IEEE, 2008. 7

[11] G. Simon, A. W. Fitzgibbon, and A. Zisserman, "Markerless tracking using planar structures in the scene," in *Augmented Reality, 2000.(ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pp. 120–128, IEEE, 2000. 8

[12] C. Harris and M. Stephens, "A combined corner and edge detector.," in *Alvey vision conference*, vol. 15, pp. 10–5244, Citeseer, 1988. 10

[13] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover.," tech. rep., DTIC Document, 1980. 10

[14] T. Lindeberg, "Scale-space theory: A basic tool for analyzing structures at different scales," *Journal of applied statistics*, vol. 21, no. 1-2, pp. 225–270, 1994. 10

[15] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984. 10

[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004. 10

[17] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer vision–ECCV 2006*, pp. 404–417, 2006. 10

[18] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer vision–ECCV 2006*, pp. 430–443, 2006. 11

[19] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," *Computer Vision–ECCV 2010*, pp. 778–792, 2010. 11

[20] V. N. Vapnik and V. Vapnik, *Statistical learning theory*, vol. 1. Wiley New York, 1998. 11, 19

[21] H.-K. Wu, S. W.-Y. Lee, H.-Y. Chang, and J.-C. Liang, "Current status, opportunities and challenges of augmented reality in education," *Computers & Education*, vol. 62, pp. 41–49, 2013. 15

[22] Itseez, "Open source computer vision library." https://github.com/itseez/opencv, 2015. 21

[23] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. 21